

关注微信公众号: PMvideo

【Spring学习34】Spring事务(4): 事务属性之7种传播行为

作者: soonfly (/authorarticle.html?author=soonfly) 2017-04-20 ☆ 收录到我的专题 (/select.html?articleId=401734)

标签 事务 (<http://www.bijishequ.com/info/search.html?searchText=事务>) methodB (<http://www.bijishequ.com/info/search.html?searchText=methodB>) PROPAGATION (<http://www.bijishequ.com/info/search.html?searchText=PROPAGATION>) 调用 (<http://www.bijishequ.com/info/search.html?searchText=调用>) 回滚 (<http://www.bijishequ.com/info/search.html?searchText=回滚>)

事务传播行为

什么叫事务传播行为? 听起来挺高端的, 其实很简单。
即使是传播, 那么至少有两个东西, 才可以发生传播。单体不存在传播这个行为。

事务传播行为 (propagation behavior) 指的就是当一个事务方法被另一个事务方法调用时, 这个事务方法应该如何进行。
例如: methodA事务方法调用methodB事务方法时, methodB是继续在调用者methodA的事务中运行呢, 还是为自己开启一个新事务运行, 这就是由methodB的事务传播行为决定的。

Spring定义了七种传播行为:

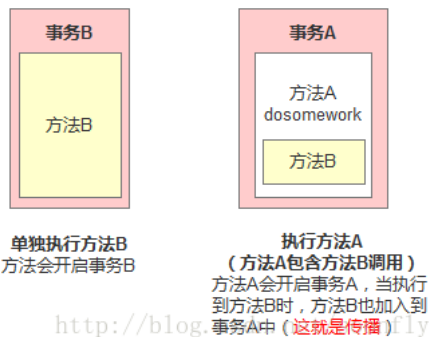
传播行为	含义
PROPAGATION_REQUIRED	表示当前方法必须运行在事务中。如果当前事务存在, 方法将会在该事务中运行。否则, 会启动一个新的事务
PROPAGATION_SUPPORTS	表示当前方法不需要事务上下文, 但是如果存在当前事务的话, 那么该方法会在这个事务中运行
PROPAGATION_MANDATORY	表示该方法必须在事务中运行, 如果当前事务不存在, 则会抛出一个异常
PROPAGATION_REQUIRED_NEW	表示当前方法必须运行在它自己的事务中。一个新的事务将被启动。如果存在当前事务, 在该方法执行期间, 当前事务会被挂起。如果使用JTATransactionManager的话, 则需要访问TransactionManager
PROPAGATION_NOT_SUPPORTED	表示该方法不应该运行在事务中。如果存在当前事务, 在该方法运行期间, 当前事务将被挂起。如果使用JTATransactionManager的话, 则需要访问TransactionManager
PROPAGATION_NEVER	表示当前方法不应该运行在事务上下文中。如果当前正有一个事务在运行, 则会抛出异常
PROPAGATION_NESTED	表示如果当前已经存在一个事务, 那么该方法将会嵌套事务中运行。嵌套的事务可以独立于当前事务进行单独地提交或回滚。如果当前事务不存在, 那么其行为与PROPAGATION_REQUIRED一样。注意各厂商对这种传播行为的支持是有所差异的。可以参考资源管理器的文档来确认它们是否支持嵌套事务

现在来看看传播行为

1、PROPAGATION_REQUIRED

如果存在一个事务, 则支持当前事务。如果没有事务则开启一个新的事务。
可以把事务想像成一个胶囊, 在这个场景下方法B用的是方法A产生的胶囊 (事务)。

PROPAGATION_REQUIRED



举例有两个方法:

```
1  @Transactional(propagation = Propagation.REQUIRED)
2  public void methodA() {
3      methodB();
4      // do something
5  }
6
7  @Transactional(propagation = Propagation.REQUIRED)
8  public void methodB() {
9      // do something
10 }
```

单独调用methodB方法时，因为当前上下文不存在事务，所以会开启一个新的事务。
调用methodA方法时，因为当前上下文不存在事务，所以会开启一个新的事务。当执行到methodB时，methodB发现当前上下文有事务，因此就加入到当前事务中来。

2、PROPAGATION_SUPPORTS

如果存在一个事务，支持当前事务。如果没有事务，则非事务的执行。但是对于事务同步的事务管理器，PROPAGATION_SUPPORTS与不使用事务有少许不同。
举例有两个方法:

```
1  @Transactional(propagation = Propagation.REQUIRED)
2  public void methodA() {
3      methodB();
4      // do something
5  }
6
7  // 事务属性为SUPPORTS
8  @Transactional(propagation = Propagation.SUPPORTS)
9  public void methodB() {
10     // do something
11 }
```

单纯的调用methodB时，methodB方法是非事务的执行的。当调用methodA时,methodB则加入了methodA的事务中,事务地执行。

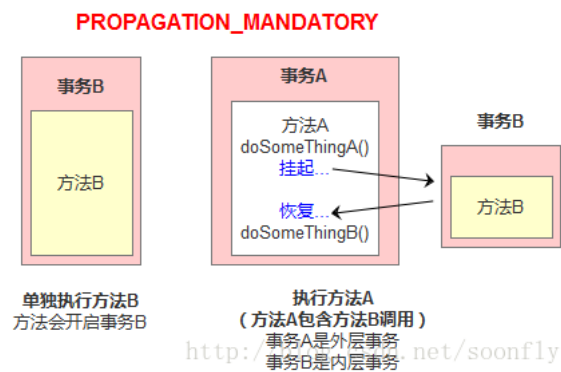
3、PROPAGATION_MANDATORY

如果已经存在一个事务，支持当前事务。如果没有一个活动的事务，则抛出异常。

```
1  @Transactional(propagation = Propagation.REQUIRED)
2  public void methodA() {
3      methodB();
4      // do something
5  }
6
7  // 事务属性为MANDATORY
8  @Transactional(propagation = Propagation.MANDATORY)
9  public void methodB() {
10     // do something
11 }
```

当单独调用methodB时，因为当前没有一个活动的事务，则会抛出异常throw new IllegalStateException(“Transaction propagation ‘mandatory’ but no existing transaction found”);当调用methodA时，methodB则加入到methodA的事务中，事务地执行。

4、PROPAGATION_MANDATORY



使用PROPAGATION_REQUIRES_NEW,需要使用 JtaTransactionManager作为事务管理器。
它会开启一个新的事务。如果一个事务已经存在，则先将这个存在的事务挂起。

```
1 @Transactional(propagation = Propagation.REQUIRED)
2 public void methodA() {
3     doSomethingA();
4     methodB();
5     doSomethingB();
6     // do something else
7 }
8
9
10 // 事务属性为REQUIRES_NEW
11 @Transactional(propagation = Propagation.REQUIRES_NEW)
12 public void methodB() {
13     // do something
14 }
```

当调用

```
1 main{ methodA(); }
```

相当于调用

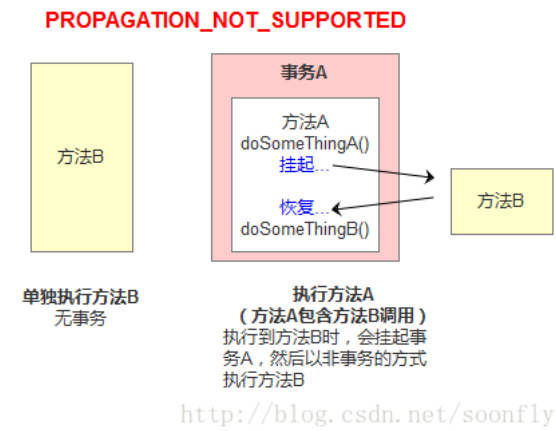
```
1 main(){
2     TransactionManager tm = null;
3     try{
4         //获得一个JTA事务管理器
5         tm = getTransactionManager();
6         tm.begin();//开启一个新的事务
7         Transaction ts1 = tm.getTransaction();
8         doSomething();
9         tm.suspend();//挂起当前事务
10    }
11    try{
12        tm.begin();//重新开启第二个事务
13        Transaction ts2 = tm.getTransaction();
14        methodB();
15        ts2.commit();//提交第二个事务
16    } catch(RuntimeException ex) {
17        ts2.rollback();//回滚第二个事务
18    } finally {
19        //释放资源
20    }
21    //methodB执行完后，恢复第一个事务
22    tm.resume(ts1);
23    doSomethingB();
24    ts1.commit();//提交第一个事务
25    } catch(RuntimeException ex) {
26        ts1.rollback();//回滚第一个事务
27    } finally {
28        //释放资源
29    }
30 }
```

1/5/2018 【Spring学习34】 Spring事务(4): 事务属性之7种传播行为--Method,su,Action,Point,事务,methodB,PROPAGATION,调用,回滚

在这里，我把ts1称为外层事务，ts2称为内层事务。从上面的代码可以看出，ts2与ts1是两个独立的事务，互不相干。Ts2是否成功并不依赖于ts1。如果methodA方法在调用methodB方法后的doSomethingB方法失败了，而methodB方法所做的结果依然被提交。而除了methodB之外的其它代码导致的结果却被回滚了

5、PROPAGATION_NOT_SUPPORTED

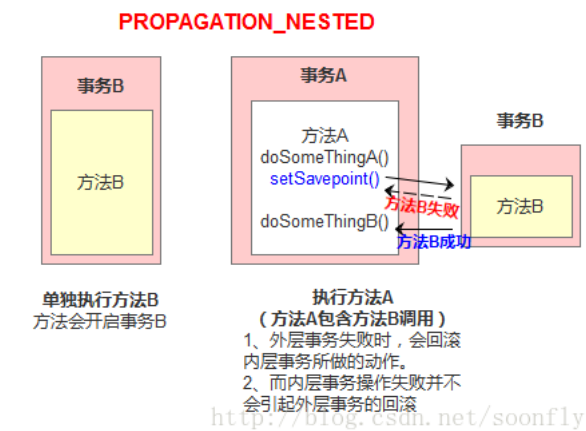
PROPAGATION_NOT_SUPPORTED 总是非事务地执行，并挂起任何存在的事务。使用PROPAGATION_NOT_SUPPORTED,也需要使用JtaTransactionManager作为事务管理器。



6、PROPAGATION_NEVER

总是非事务地执行，如果存在一个活动事务，则抛出异常。

7、PROPAGATION_NESTED



如果一个活动的事务存在，则运行在一个嵌套的事务中。如果没有活动事务,则按TransactionDefinition.PROPAGATION_REQUIRED 属性执行。

这是一个嵌套事务,使用JDBC 3.0驱动时,仅仅支持DataSourceTransactionManager作为事务管理器。

需要JDBC 驱动的java.sql.Savepoint类。使用PROPAGATION_NESTED，还需要把PlatformTransactionManager的nestedTransactionAllowed属性设为true(属性值默认为false)。

这里关键是嵌套执行。

```
1  @Transactional(propagation = Propagation.REQUIRED) methodA(){
2  doSomethingA();
3  methodB();
4  doSomethingB();
5  }
6
7  @Transactional(propagation = Propagation.NESTED) methodB(){
8  .....
9  }
```

如果单独调用methodB方法，则按REQUIRED属性执行。如果调用methodA方法，相当于下面的效果：

```
1  main(){
2  Connection con = null;
3  Savepoint savepoint = null;
4  try{
5  con = getConnection();
6  con.setAutoCommit(false);
7  doSomethingA();
8  savepoint = con2.setSavepoint();
9  try{
10 methodB();
11 } catch(RuntimeException ex) {
12 con.rollback(savepoint);
13 } finally {
14 //释放资源
15 }
16 doSomethingB();
17 con.commit();
18 } catch(RuntimeException ex) {
19 con.rollback();
20 } finally {
21 //释放资源
22 }
23 }
```

当methodB方法调用之前，调用setSavepoint方法，保存当前的状态到savepoint。如果methodB方法调用失败，则恢复到之前保存的状态。但是需要注意的是，这时的事务并没有进行提交，如果后续的代码(doSomethingB()方法)调用失败，则回滚包括methodB方法的所有操作。嵌套事务一个非常重要的概念就是内层事务依赖于外层事务。外层事务失败时，会回滚内层事务所做的动作。而内层事务操作失败并不会引起外层事务的回滚。

PROPAGATION_NESTED 与PROPAGATION_REQUIRES_NEW的区别:

它们非常类似,都像一个嵌套事务，如果不存在一个活动的事务，都会开启一个新的事务。
使用 PROPAGATION_REQUIRES_NEW时，内层事务与外层事务就像两个独立的事务一样，一旦内层事务进行了提交后，外层事务不能对其进行回滚。两个事务互不影响。两个事务不是一个真正的嵌套事务。同时它需要JTA事务管理器的支持。

使用PROPAGATION_NESTED时，外层事务的回滚可以引起内层事务的回滚。而内层事务的异常并不会导致外层事务的回滚，它是一个真正的嵌套事务。DataSourceTransactionManager使用savepoint支持PROPAGATION_NESTED时，需要JDBC 3.0以上驱动及1.4以上的JDK版本支持。其它的JTATrasactionManager实现可能有不同的支持方式。

PROPAGATION_REQUIRES_NEW 启动一个新的,不依赖于环境的“内部”事务.这个事务将被完全 committed 或 rolled back 而不依赖于外部事务,它拥有自己的隔离范围,自己的锁,等等.当内部事务开始执行时,外部事务将被挂起,内务事务结束时,外部事务将继续执行。

另一方面,PROPAGATION_NESTED 开始一个“嵌套的”事务,它是已经存在事务的一个真正的子事务.潜套事务开始执行时,它将取得一个 savepoint.如果这个嵌套事务失败,我们将回滚到此 savepoint.潜套事务是外部事务的一部分,只有外部事务结束后它才会被提交。

由此可见,PROPAGATION_REQUIRES_NEW 和 PROPAGATION_NESTED 的最大区别在于,PROPAGATION_REQUIRES_NEW 完全是一个新的事务,而 PROPAGATION_NESTED 则是外部事务的子事务,如果外部事务 commit,嵌套事务也会被 commit,这个规则同样适用于 roll back.




登录

(/login.htm?redirectUrl=%2Fdetail/401734?pn=2)

0条评论或问题

社区邀请

笔记社区是一个面向中高端IT开发者、程序员的知识共享社区，通过网络抓取与文章分类总结，由专家为用户提供高质量的专题文章系列。

 邀请您成为社区专家 >> (<http://www.bijishequ.com/creat.html>)

后发表评论

原文链接：<http://blog.csdn.net/soonfly/article/details/70305683>

声明：所有文章资源均从网络抓取，如果侵犯到您的著作权，请联系删除文章。联系方式请关注微信公众号PMvideo【锤子视频-程序员喜欢的短视频】，或者加笔记社区开发者交流群 628286713。

相关标签

- Struts (/tag/list?tagId=1440)
- loc (/tag/list?tagId=1441)
- Spring (/tag/list?tagId=1438)
- SpringMVC (/tag/list?tagId=1560)
- LinkedList (/tag/list?tagId=2462)
- Serializable (/tag/list?tagId=24)
- FileNotFoundException (/tag/list?tagId=23)
- ByteArrayOutputStream (/tag/list?tagId=26)
- Method (/tag/list?tagId=25)
- userdel (/tag/list?tagId=1138)
- useradd (/tag/list?tagId=1139)
- usermod (/tag/list?tagId=1140)
- su (/tag/list?tagId=1142)
- groupadd (/tag/list?tagId=1141)
- PUT (/tag/list?tagId=922)
- PrivateKey (/tag/list?tagId=925)
- IntentService (/tag/list?tagId=923)
- ListSelectionModel (/tag/list?tagId=693)
- EventPriority (/tag/list?tagId=926)
- Action (/tag/list?tagId=924)
- ContentObserver (/tag/list?tagId=689)
- Session (/tag/list?tagId=918)
- OnPreferenceClickListener (/tag/list?tagId=690)
- IStructuredSelection (/tag/list?tagId=559)
- OutputKeys (/tag/list?tagId=400)
- ApplicationContextAware (/tag/list?tagId=1466)
- BaseColumns (/tag/list?tagId=403)
- PropertyChangeEvent (/tag/list?tagId=402)
- ISelectionChangedListener (/tag/list?tagId=728)
- PathParam (/tag/list?tagId=560)
- AlarmManager (/tag/list?tagId=563)
- Point (/tag/list?tagId=727)
- ArrayUtils (/tag/list?tagId=726)
- EventListener (/tag/list?tagId=399)
- Path (/tag/list?tagId=687)
- ImmutableSet (/tag/list?tagId=729)
- IWorkbenchPage (/tag/list?tagId=640)
- PatternSyntaxException (/tag/list?tagId=637)
- IResource (/tag/list?tagId=639)
- Transactional (/tag/list?tagId=638)
- Player (/tag/list?tagId=636)
- J2EE (/tag/list?tagId=2397)
- 编程语言 (/tag/list?tagId=2761)
- java (/tag/list?tagId=2396)
- Linux用户与工作组 (/tag/list?tagId=1119)
- SELinux (/tag/list?tagId=1143)
- Linux (/tag/list?tagId=1079)
- scp (/tag/list?tagId=1118)

相关文章

- Spring源码分析之AOP (/detail/577731?p=)
- 通过注解实现Spring 声明式事务管理 (/detail/577865?p=)
- Spring Data Jpa(分页、Specification、Criteria) (/detail/578235?p=)
- spring boot 集成shiro详细教程 (/detail/578006?p=)
- SpringIOC源码阅读—BeanDefinitionParserDelegate (/detail/577695?p=)
- SpringIOC源码阅读—BeanDefinitionDocumentReader (/detail/577693?p=)
- 使用 Spring Boot Starter 封装 微信 SDK (/detail/578400?p=)
- Spring BeanPostProcessor接口使用 (/detail/577929?p=)
- Spring Boot 与 I18N (/detail/578182?p=)
- Spring Aware相关接口的使用 (/detail/577831?p=)

相关专题



springBoot RabbitMQ

全面介绍springBoot RabbitMQ

共 0 篇文章

(/subject/336)



Springboot详细入门教程

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。

共 14 篇文章

(/subject/179)

